さて、前回のレッスンでは、Dartストリームと、それらがいつでも更新される可能性のあるデータをリッスンできるようにする方法について説明しました。そして、新しいデータが届くと、いくつかのコードがトリガーされ、その新しいデータにアクセスできるようになります。

しかし、今度はFlutterランドに戻って、これらのストリームから返されたこのデータを実際のウィジェットに変換する必要があります。

つまり、Firebaseコレクション内のデータを取得して、チャット画面の一部のウィジェット内に表示する方法を見つける必要があります。

それでは、どうすればこれを達成できますか？

それでは、2つのステップでこれに取り組みましょう。

ステップ1は、できるだけ単純な方法でデータを表示するだけです。

そして、ステップ2は、見栄えを良くすることです。

そして、このレッスンでは、データを表示するだけのステップ1に焦点を当てます。

そして、チャットメッセージを表示する最も簡単な方法は、列内にすべて表示することです。

そのため、空想を行う前に、テキストウィジェットのリストを子として含む列を作成して、チャットメッセージを表示するだけです。

これが私たちのアプリが今どのように見えるかです。そして、ここに私たちが目指しているものがあります。

これは、テキストウィジェットのその列を含む画面がレッスンの終わりまでにどのように見えるかです。しかし、ここにあるのは、Dartのスナップショットメソッドです。

そのため、ストリームを処理でき、テキストウィジェットのリストを作成できるものが必要です。

ただし、ウィジェットを作成するだけでなく、新しいチャットメッセージがストリームに着信するたびにウィジェットを更新する必要があります。

それではどうすればそれができますか？

ストリームビルダーと呼ばれるウィジェットのヘルプを使用できます。

これは、新しいデータが届くたびに、データのスナップショットを実際のウィジェットに変えるものです。そのため、「ストリーム」から新しいデータが送信されるたびに再構築でき、設定状態を使用してそれを行います。

つまり、ストリームに新しい値があるたびに、セット状態が呼び出されます。このコラムの内側にあるように、現在、子のリスト内には子が1つしかありません。それでは、もう1つ追加しましょう。

そして、メイン軸をspaceBetweenに設定するように列を設定しているため、ここに複数のアイテムがある場合、この送信テキストフィールドとボタンを下に押して、メッセージを表示します頂点で。

そして、これらのメッセージをストリームビルダーを使用して表示します。ここにストリームビルダーがあり、他のウィジェットを使用するのと同じように使用できます。

そのため、それは私たちのコラムに入り、私たちのストリームビルダーには本当に重要な2つのプロパティがあります。

1つはストリームです。データはどこから来るのでしょうか。

これは、ここで行ったことと同じです。

ストリームのこの部分をコピーすることも、最初から入力することもできます。

そのため、firestore.collectionを使用してメッセージコレクションをタップし、ストリームであるこれらのスナップショットを取得します。特に、クエリスナップショットのストリームです。

クエリスナップショットはFirebaseのクラスであり、最終的にはチャットメッセージが含まれます。これで、ストリームビルダーをサブスクライブするストリームができたので、新しいデータがいつ再構築されるかがわかります。

さて、次に提供しなければならないのはビルダーです。

ビルダーは奇妙なものです。

クイックドキュメントを読むと、ビルド戦略と呼ばれるものを提供する必要があることがわかります。

つまり、ストリームビルダーが実際に行うべきことのロジックを提供する必要があります。

どうして？

ストリームビルダーがストリームと対話しており、チャットメッセージのような新しいイベントが送信されるたびに、ストリームビルダーがスナップショットを受信することを忘れないでください。この時点で、ビルダー関数は画面に表示されるメッセージのリストを更新する必要があります。

言い換えると、ビルダーは、ストリームビルダーのすべての子、つまり、前述したテキストウィジェットの列を再構築する必要があります。

では、クイックドキュメントで見ることができるこのスナップショットは何ですか？

タイプが非同期スナップショットであることがわかります。

この非同期スナップショットは、ストリームとの最新の対話を表します。チャットメッセージはこの非同期スナップショットのどこかに埋まっているため、ビルダー機能を使用してアクセスできます。

それでは、ビルダーを詳しく見てみましょう。また、ビルダーは匿名のコールバックを取得するものであり、2つの入力があります。そのため、コンテキストとスナップショットを渡すコールバックをトリガーし、実際のウィジェットを返します。

この場合、ビルダー関数が返すウィジェットは、テキストウィジェットを含む列になります。先に進み、その匿名関数を作成してみましょう。最初に渡すのは「コンテキスト」です。

2つ目は、取得したスナップショットです。これで、これらの中括弧内に、ビルドされたコンテキストと、ストリームビルダーが存在するスナップショットの両方にアクセスできるようになります。

このスナップショットは、ここまでのスナップショットとは異なります。

Firebaseのクエリスナップショットを扱っていました。

しかし、ストリームビルダーで作業しているため、ここにFlutterの非同期スナップショットがあります。

ただし、非同期スナップショットには、実際にはFirebaseからのクエリスナップショットが含まれています。

私の言いたいことをお見せしましょう。

非同期スナップショットからチャットメッセージを掘り始めましょう。

したがって、ウィジェットをレンダリングするために使用する前に最初に確認したいのは、

空のスナップショットを取得しませんでした。

そのため、スナップショットをタップして.hasDataを呼び出すことで確認できます。

そのため、これは、このスナップショットにデータがあるか、nullであるかを示すブール値を返します。

そして、データがある場合は、そのデータを使用します。

そこで、messagesという最終変数を作成し、それをsnapshot.dataと等しくなるように設定します。

これが、非同期スナップショット内のデータにアクセスする方法です。これは非同期スナップショットのドキュメントで見ることができます。非同期スナップショットのドキュメントでは、「現在アクセスしているdataプロパティと、ストリームのスナップショットがnullでないことを確認するために使用しているhasDataプロパティの両方を確認できます。firebaseのクエリスナップショットはデータプロパティ内に保存されます。

ただし、Android Studioでコードを振り返ると、このクエリスナップショットデータ型は表示されません。

代わりに、データ型が動的であることがわかります。その理由は、このストリームビルダーを構築したにもかかわらず、どのタイプのストリームを取得するかを通知しなかったためです。

一方、スナップショットを見ると、クエリスナップショットのストリームが返されることがわかります。

これは、FirebaseクラウドFirestoreからのデータ型であり、メッセージコレクション内のすべてのデータを表すものです。したがって、Stream Builderにそのタイプを追加できます。したがって、ストリームビルダーを構築しており、クエリスナップショットをリッスンしていることがわかります。

それが最終的にストリーム内にあるデータです。

そして、一度、どのような種類のデータがあるかを伝えると、データオブジェクトが実際のクエリスナップショットに更新されます。つまり、含まれるドキュメントなど、他にどんなものがあるかがわかります。

ここで何をしたかを考えてみましょう。

非同期スナップショットには、Firebaseからのクエリスナップショットが含まれています。

dataプロパティを介してクエリスナップショットにアクセスします。

クエリスナップショットオブジェクトを処理しているため、documentsプロパティなどのクエリスナップショットのプロパティを使用できます。

これにより、ドキュメントのスナップショットのリストが表示されます。

要約すると、ストリームビルダーの非同期スナップショットには、Firebaseのクエリスナップショットが含まれています。クエリスナップショットには、ドキュメントスナップショットのリストが含まれます。今は2層になっていますが、チャットメッセージのテキストを取得するにはさらに深く掘り下げる必要があります。それでは、どのようにして単一の

メッセージテキストを含むドキュメント？

さて、今、ドキュメントのリストがあります。そのために、信頼できる古い友人であるFORループを使用します。そして、私はFORループを使用して、大量のテキストウィジェットを作成します。

それでは、テキストウィジェットのリストを作成しましょう。私はそれを「messageWidgets」と呼び、最初から空のリストに設定します。

次の部分はforループを使用することです。

したがって、メッセージ内のvarメッセージの場合。これがドキュメントのスナップショットのリストです。このFORループ内で、「message.dataをタップしてメッセージテキストをタップします。ここでデータは印刷したものです」と、これは文字列で構成されるキーのマップであり、値は動的になります。

これが、単一のドキュメントスナップショットに含まれるデータです。

各ドキュメントには、チャットメッセージテキストと送信者の2つの情報が含まれていることに注意してください。

コードを振り返ってみると、この.dataプロパティがこことここの2つの場所にあるにもかかわらず、2つの非常に異なるタイプのオブジェクトを扱っていることをすぐに指摘する価値があります。

どちらもたまたまdataというプロパティを持っています。スナップショットはFlutterからの非同期スナップショットであり、メッセージはFirebaseからのドキュメントスナップショットです。

コンソール内を見ると、これらはそれぞれスナップショットデータであることがわかります。

そして、キーはこれであり、フィールドと呼ばれるものは何でも、値はそれが含むもの、つまりテキストです。そのため、メッセージテキストはデータベースのtextというフィールドの下に保存されます。

そのため、キーとして文字列としてそれを入力し、ここで正確に同じことを確認する必要があります。

したがって、タイプミスや大文字の変更はありません。

メッセージスナップショットから取り出す必要がある他の変数は、メッセージ送信者です。そして、これは「message.dataと等しくなります」であり、送信者キーの下の値を利用します。

メッセージテキストとメッセージ送信者ができたので、最終的にメッセージウィジェットを作成します。これは、データを含むテキストウィジェットになります。つまり、メッセージテキストとメッセージはメッセージ送信者から送信されます。

だから、messageSender。そして、addメソッドを使用して、メッセージウィジェットのリストにメッセージウィジェットを追加する準備ができました。

したがって、FORループの実行が完了したら、いよいよストリームビルダーの出力としてウィジェットを実際に返す時間になります。

そして、私は列を返すつもりです。なぜなら、それは子供の全体の束を含み、列の子供はテキストウィジェットのリストになり、それぞれがメッセージと送信者を持っているからです。

それが私がここに入れようとしているものです。

そして今、セミコロンでそれを閉じたら、保存を押すことができます。そして、これまでに作成したすべてのメッセージをチャット画面で見ることができます。

そして、メイン列のMainAxisAlignmentをspaceBetweenとして取得しているため、そのテキストフィールドを下にプッシュし、テキストウィジェットの列を右上にプッシュします。

この時点でFirebase Cloud Firestoreに新しいメッセージを送信すると、データベースに到達した瞬間に、新しいメッセージが送信されてチャット画面に直接追加されることも通知されます。

チャット画面に直接追加されるのはなぜですか？

ストリームビルダーが新しい非同期スナップショットを受け取るため、これによりビルダー機能がトリガーされ、画面上のテキストウィジェットのリストが再構築されます。

つまり、コードはチャットメッセージの送信などの新しいイベントに反応しています。

ちょっと沈めましょう。

これは本当にすてきな方法だと思います。

しかし、コードを見てみると、実際に処理していないのは、スナップショットにデータがない場合はどうでしょうか？

その場合、Stream Builderに対して何を出力しますか？

これについて少し考えてみましょう。

どの場合、スナップショットにデータがないのでしょうか？

Firebaseに接続する前に、最初のスナップショットにnull値がある場合があります。

それでは、この状況に対処しましょう。

最初のスナップショットにデータがなく、Firebaseにまだ接続していない場合、私がやりたいことは、ユーザーがアプリを待つことを知っているように、何らかのスピナーまたは進行状況インジケーターを表示することです。スナップショットにデータがあるかどうかを確認してこれらのことを実行する代わりに、実際にその頭をひっくり返します。そのため、スナップショットの閉じ括弧にデータがあることを確認し、同じ行で閉じます。代わりに、スナップショットにデータがないかどうかを確認します。

したがって、この方法では、まだデータを取得していないかどうかを確認できます。この場合、別のウィジェットを返します。

センターウィジェットを返し、センターウィジェット内でCircularProgress Indicatorを使用します。

円形の進行状況インジケーターは、モーダル進行状況HUDの基になっています。

前のレッスンでログインおよび登録した後、プログレススピナーを実装したことを思い出してください。

モーダルプログレスHUDを使用すると、showSpinnerと呼ばれるブール値を使用して、スピンするタイミングとスピンを停止するタイミングを簡単に指定できました。

ただし、この場合、実際に必要なのは、単純な循環進行インジケーターです。

背景色を指定するか、水色のアクセントに変更することもできますが、これは、データを取得するとすぐにストリームビルダーが実際にデータを取得するため、実際にデータがまだない場合のみです代わりに列を作成します。

そのため、これはデータがないときにのみ表示およびスピンします。

そして、データを取得するとすぐに破棄されます。現時点では、非常にquiteいメッセージのリストが表示されています。そして、それは素晴らしいチャットアプリのようには見えません。

このようなものを実際に使用する人はいません。

そのため、次のレッスンでは、メッセージをバブルにして、Flutterリストビューについても学習することで、本物のメッセージングアプリのように見せます。

それで、それ以上については、次のレッスンでお会いしましょう。